Ministry of Electronics & Information Technology

सुस्वागतम्
நல்வரவு
ಸುಸ್ವಾಗತಂ
സുസ്വാഗതം
సుస్వాగతం
ସୁସ୍ୱାଗତମ
ਸੁਆਗਤਮ
سوسواگتم
خوش آمدید

NPSF User Workshop 2023

24 July – 24 Aug

Presentation On

PEFT For LLM Using NeMo

Smritika Sadhukhan – Quantiphi

8th August

National PARAM Supercomputing Facility

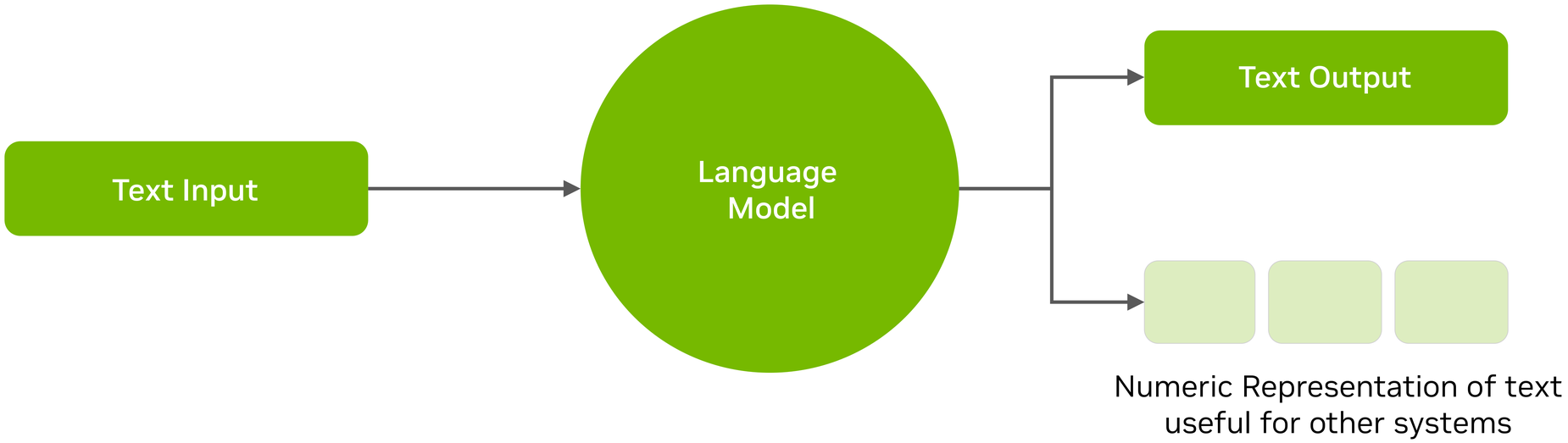
AGENDA

- Overview of Large Language Models, and GPT (with comparison of GPT1 GPT2 and GPT3)
- NVIDIA NeMo
- PEFT
  - P-Tuning
  - Prompt Tuning
  - Adapter
  - LoRA
- Finetuning
  - IFT/SFT
  - NeMo RLHF
  - NeMo Guardrails
- P-Tuning
  - P-Tuning
  - P-tuning Demo

Overview of large language models, and GPT (with comparison of GPT1 GPT2 and GPT3)

Large Language Models(LLMs)

- A large language model is a type of machine learning model that is trained on a large corpus of text data to generate outputs for various natural language processing (NLP) tasks, such as text generation, question answering, and machine translation.
- Large language models are typically based on deep learning neural networks such as the Transformer architecture and are trained on massive amounts of text data from various data sources, often involving billions of words which allows them to generate output for many tasks.



Use Cases of LLMs

Large language models can be applied to a variety of use cases and industries, including healthcare, retail, tech, and more. The following are use cases that exist in all industries:

Text generation

Sentiment analysis

Chatbots

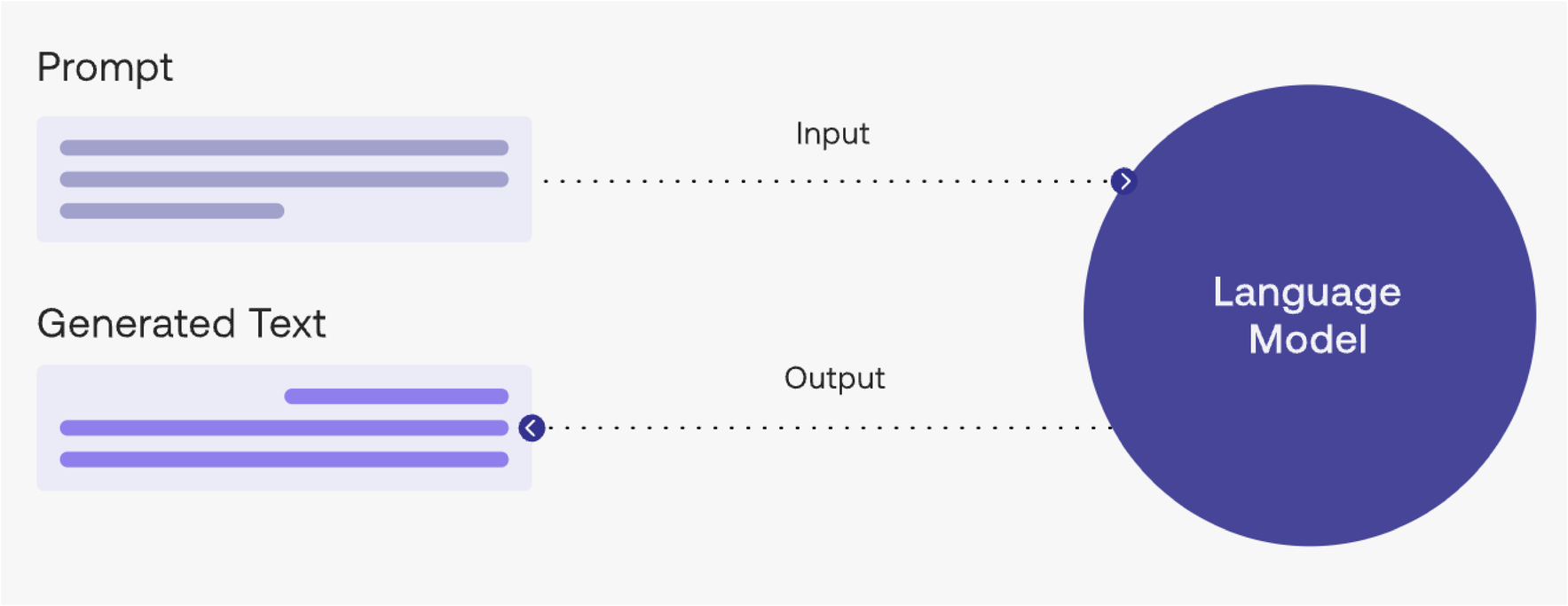
Textual Entailment Recognition

Question Answering

Code generation

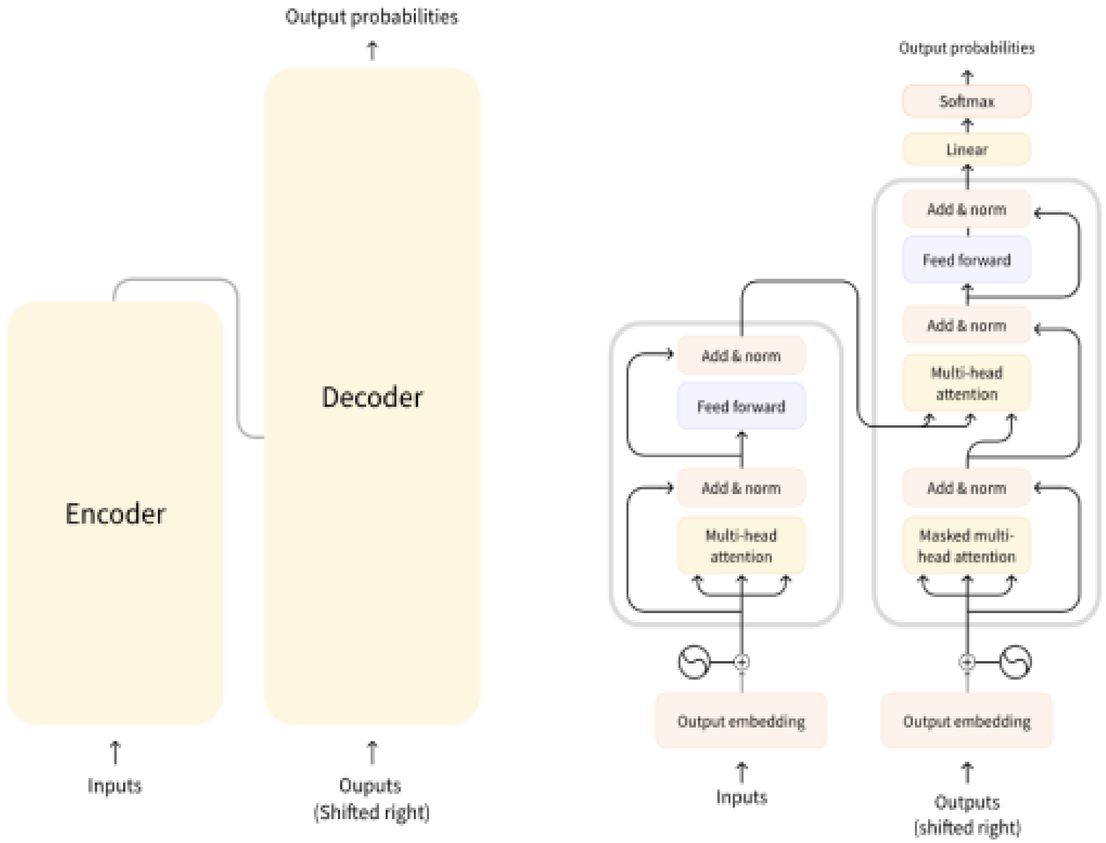
What is Prompt Engineering?

A prompt can contain information like the *instruction* or *question* you are passing to the model and include other details such as *context*, *inputs*, or *examples*.



GPT

Transformer Architecture



Architecture

- A transformer is a deep learning model that adopts the mechanism of self-attention, differentially weighting the significance of each part of the input data. It is used primarily in the Fields of natural language processing (NLP) and computer vision (CV).
- The Transformer model was introduced in Attention Is All You Need and improved in Scaling Neural Machine Translation. This implementation is based on the optimized implementation in Facebook's Fairseq NLP toolkit, built on top of PyTorch.
- Based on Encoder-Decoder Architecture, wherein encoder understands language, whilst decoder generates language.

Categories of Transformer Models

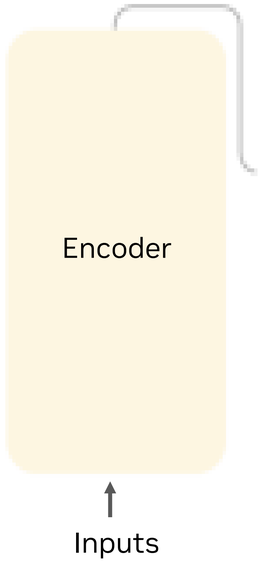
Encoders

For understanding Language

Suited for task requiring an understanding of the full sentence, such as sentence classification, named entity recognition, and extractive question answering.

Models:

- BERT
- ALBERT
- DistilBERT



Decoders

For Generative Models

Suited for tasks involving Text Generation

Models:

- GPT-3
- GPT-2

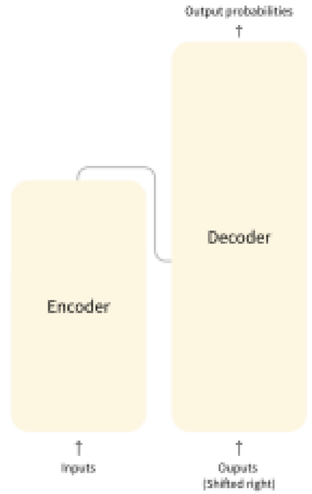
Encoders-Decoders

Sequence to Sequence

Suited for tasks around generating new sentences depending on a given input, such as summarization, translation, or generative question answering.

Models:

- T5
- Multilingual – mT5



Comparison between GPT1, GPT2 and GPT3

	GPT1	GPT2	GPT3
Release Year	2018	2019	2020
Architecture	12-level, 12-headed Transformer decoder (no encoder), followed by linear-softmax.	GPT-1, but with modified normalization	GPT-2, but with modification to allow larger scaling.
Parameter Count	0.12 billion	1.5 billion	175 billion
Training Data	BookCorpus: 4.5 GB of text, from 7000 unpublished books of various genres.	WebText: 40 GB of text, 8 million documents, from 45 million webpages upvoted on Reddit.	570 GB plaintext, 0.4 trillion tokens. Mostly CommonCrawl, WebText, English Wikipedia, and two books corpora
Notable Features	Improved language understanding and coherent text generation.	Impressive results in various language tasks, controversy due to its ability to generate human-like text.	Unprecedented results in a wide range of language tasks, ability to complete complex tasks and write code, concerns about bias and high computational costs

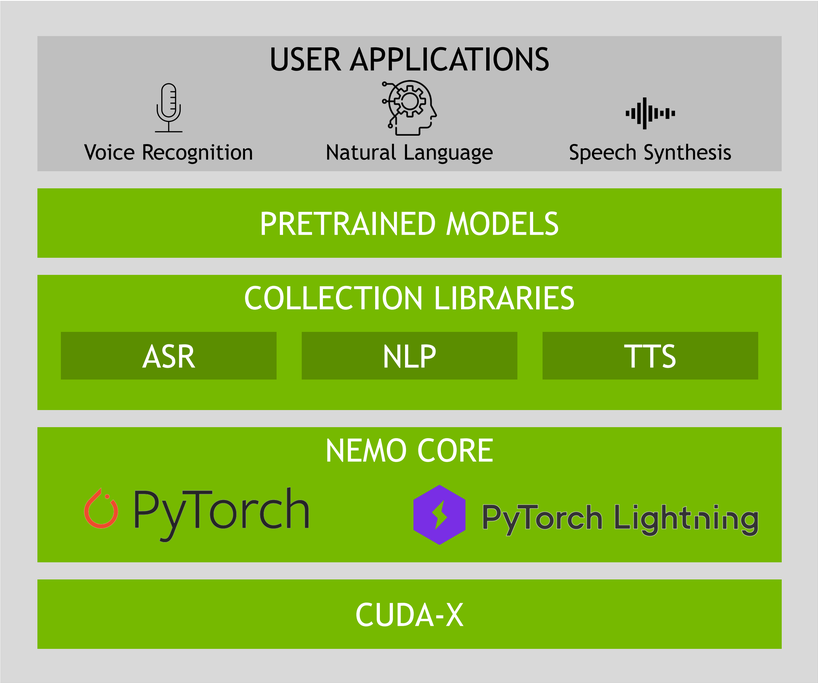
Challenges

Compute-, cost-, and time-intensive workload:	Significant capital investment and large-scale compute infrastructure are necessary to maintain and develop LLMs.
Scale of data required:	As mentioned, training a large model requires a significant amount of data. Many companies struggle to get access to large enough data.
Technical expertise:	Due to their scale, training and deploying large language models are very difficult.

NVIDIA NeMo

NVIDIA NeMo

Framework to build, customize, and deploy generative AI models with billions of parameters.



- DL-Based Speech & Language Understanding Models
- Include Semantic Checking
- Easy-to-Use APIs
- Optimized Training Performance
- 100+ Pre-Trained GPU-Optimized Checkpoints
- Scale to 1000s of NVIDIA GPUs

Rapid Model Building

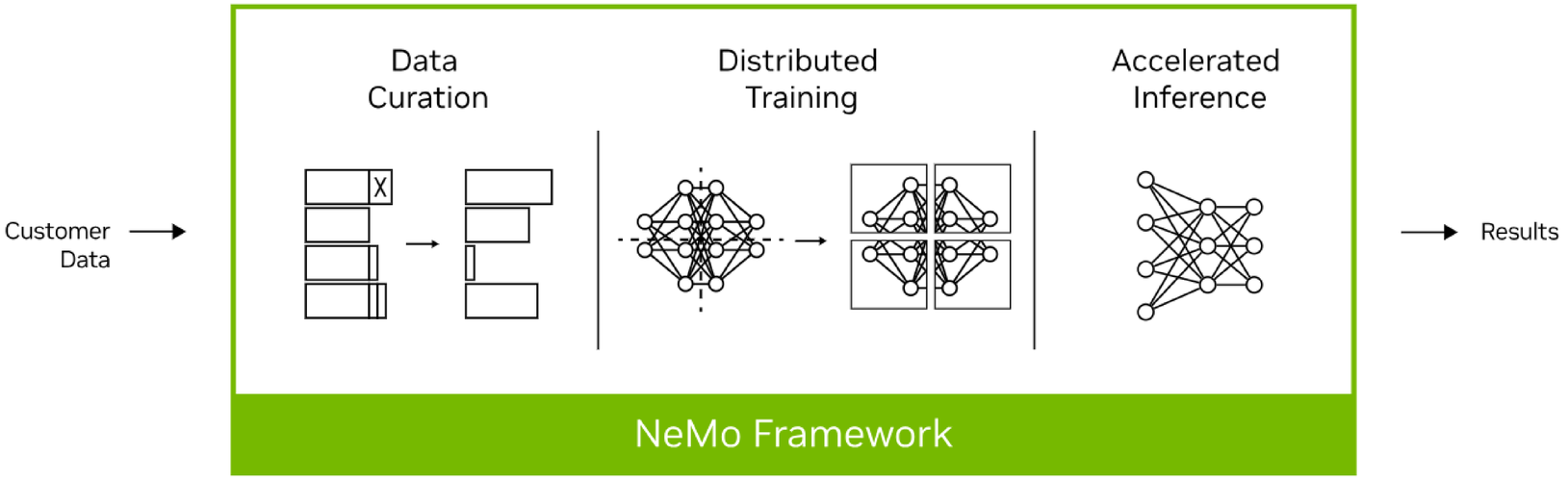
Customizable Models

Deploy with NVIDIA AI

Flexible, Open-Source, Rapidly Expanding Ecosystem

Easily Composable new Model Architectures

Introduction to NVIDIA NeMo Framework



- NeMo Megatron has an Enterprise edition which contains tools for data preprocessing, hyperparameter tuning, container, scripts for various clouds and more.
- NVIDIA NeMo is an end-to-end cloud-native enterprise framework for developers to build, customize, and deploy generative AI models with billions of parameters.
- The NeMo framework provides an accelerated workflow for training with 3D parallelism techniques.
- It offers a choice of several customization techniques and is optimized for at-scale inference of large-scale models for language and image applications, with multi-GPU and multi-node configurations.
- NeMo makes generative AI model development easy, cost-effective, and fast for enterprises.

Benefits

Fastest Training on GPUs

Easy to Use

Fully Flexible

Run in the Cloud and On Premises

Customize

Enterprise Support

- Fastest Training on GPUs: Use state-of-the-art training techniques to maximize throughput and minimize training time for foundation models with billions or trillions of parameters.
- Easy to Use: A cloud-native framework with all dependencies pre-packaged and installed with validated receipts for training language and image generative AI models to convergence and deploy for inference.
- Fully Flexible: An open-source approach offering full flexibility across the pipeline—from data processing, to training, to inference of generative AI models.
- Run in the Cloud and On Premises: Train and deploy foundation models of any size on any GPU infrastructure. Supported on all NVIDIA DGX™ systems, NVIDIA DGX Cloud™, Microsoft Azure, Oracle Cloud Infrastructure, and Amazon Web Services.
- Customize: Offers tools to customize foundation models for enterprise hyper-personalization.
- Enterprise Support: Battle hardened, tested, and verified containers built for enterprises.

https://ngc.nvidia.com/catalog/containers/nvidia:nemo

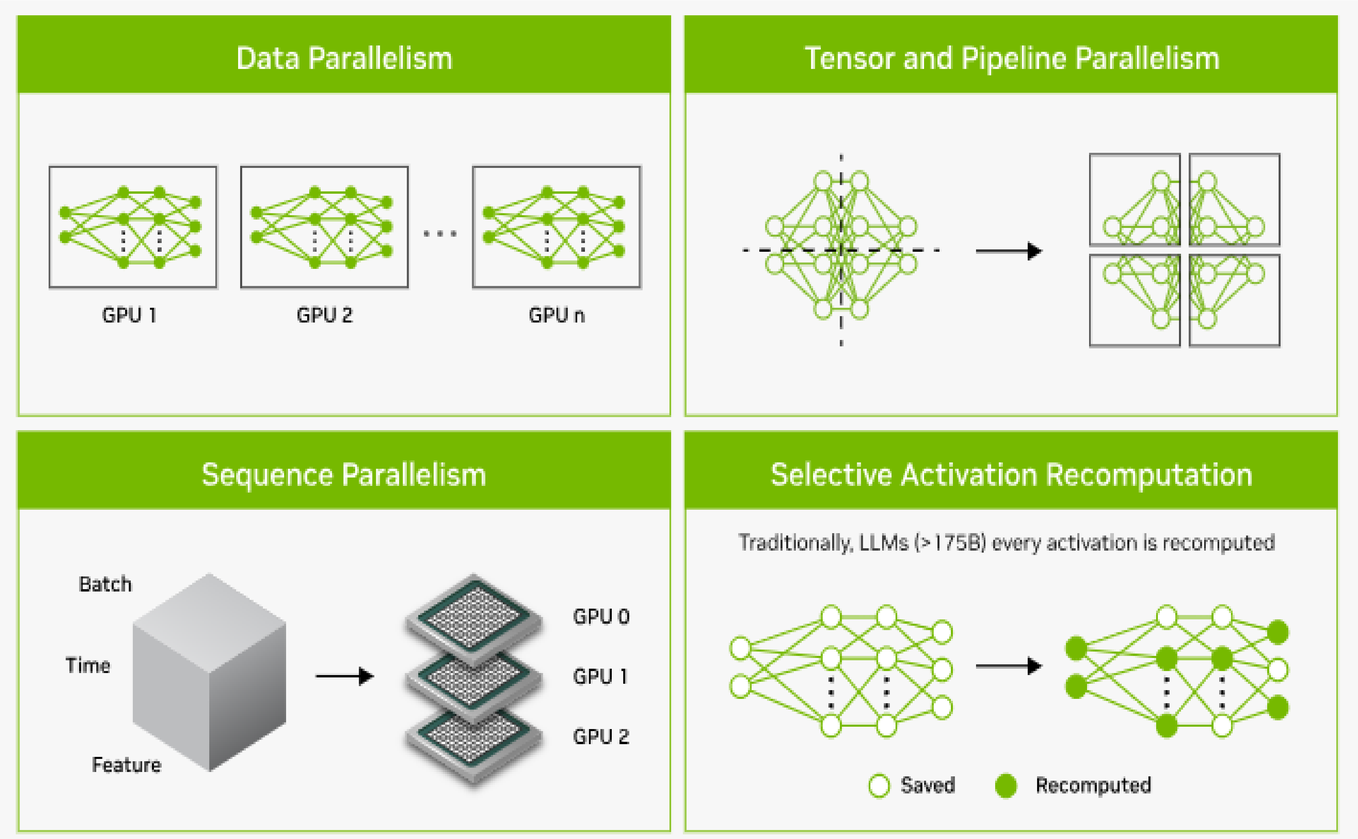
https://github.com/NVIDIA/NeMo

Key Features to Develop Large Language Models

State of the art Training Techniques

The NeMo framework uses 3D parallelism techniques:

- Tensor parallelism to scale models within nodes
- Data and pipeline parallelism to scale data and models across thousands of GPUs
- Sequence parallelism to distribute activation memory across tensor parallel devices



https://ngc.nvidia.com/catalog/containers/nvidia:nemo

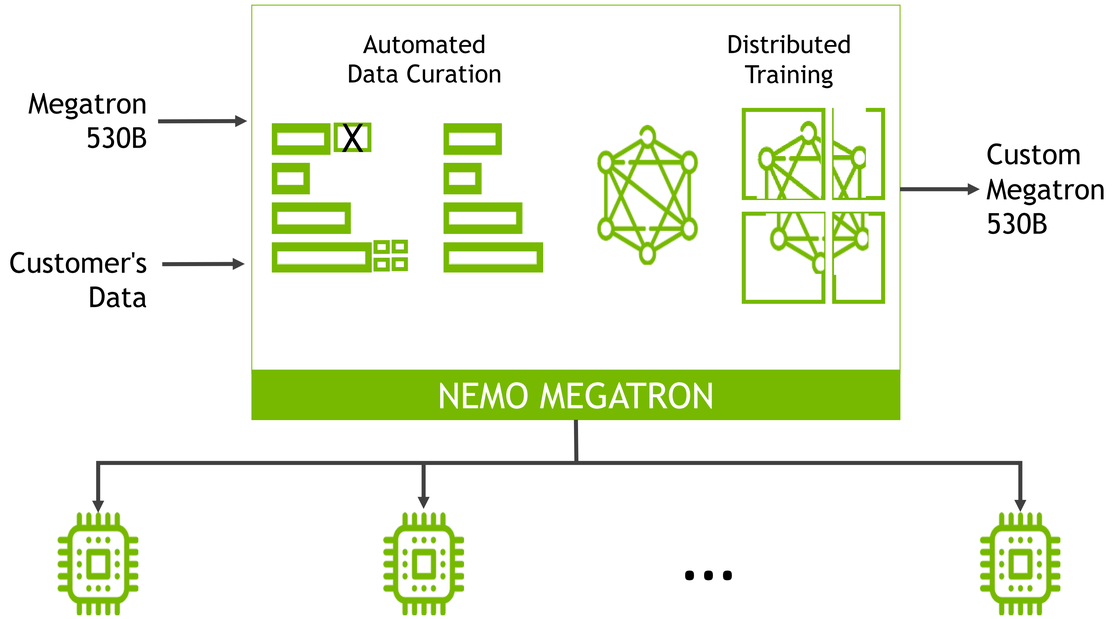
https://github.com/NVIDIA/NeMo

MEGATRON LM

Accelerated Framework For Training Large Scale NLP Models

- Scale To Models with Trillions of Parameters
- Automated Data Curation for Training
- Pipeline, Tensor & Data Parallelism
- 20B Parameter Model in 1 month on DGXSuperPod
- Optimized for DGX SuperPod

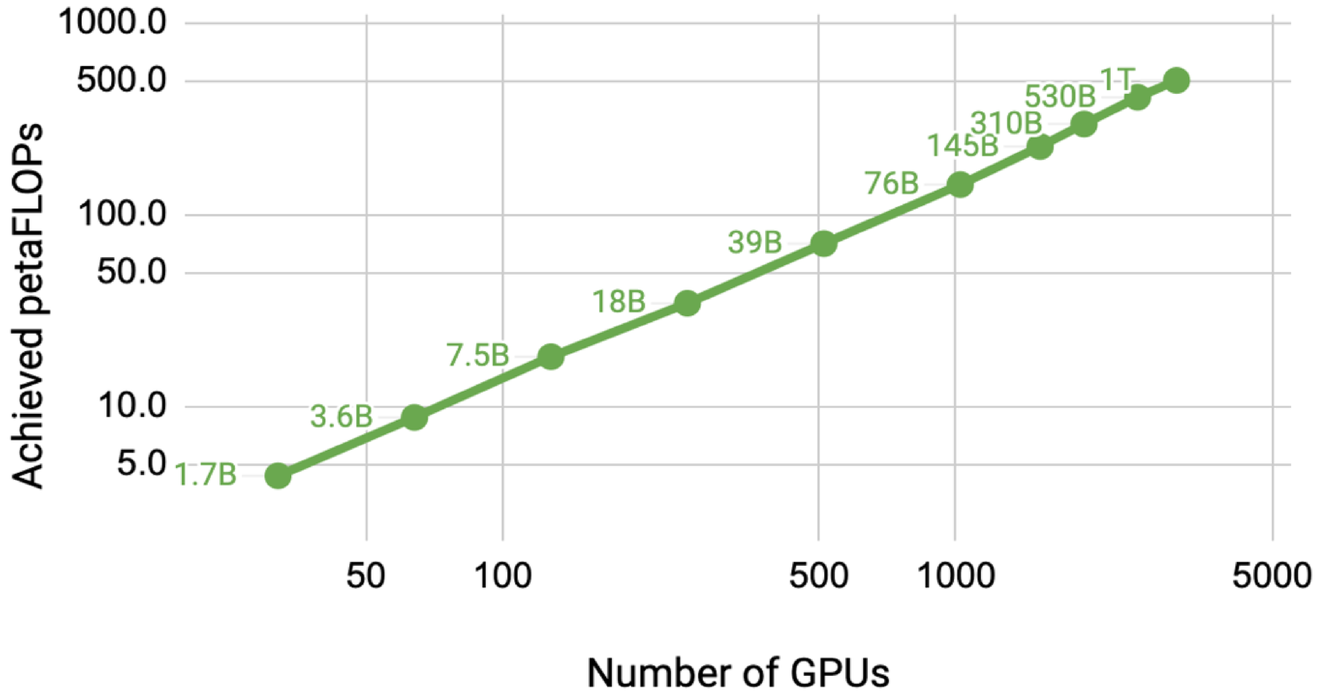
- NeMo Megatron supports several types of models:
  - GPT-style models (decoder only)
  - T5/BART/UL2-style models (encoder-decoder)
  - BERT-style models (encoder only)
  - Retro-style models (decoder only)



Sign up for Early Access

https://docs.nvidia.com/deeplearning/sdk/mixed-precision-training/index.html

Scale to Models with Trillions of Parameters



- FLOPs are measured for end-to-end training, i.e., includes all operations including data loading, optimization, and even logging.
- All models use a vocabulary size of 51,200 and a sequence length of 2048
- They vary hidden size, number of attention heads, and number of layers to arrive at a specific model size.

TRAINING BENCHMARKS

Training (GPT-3)

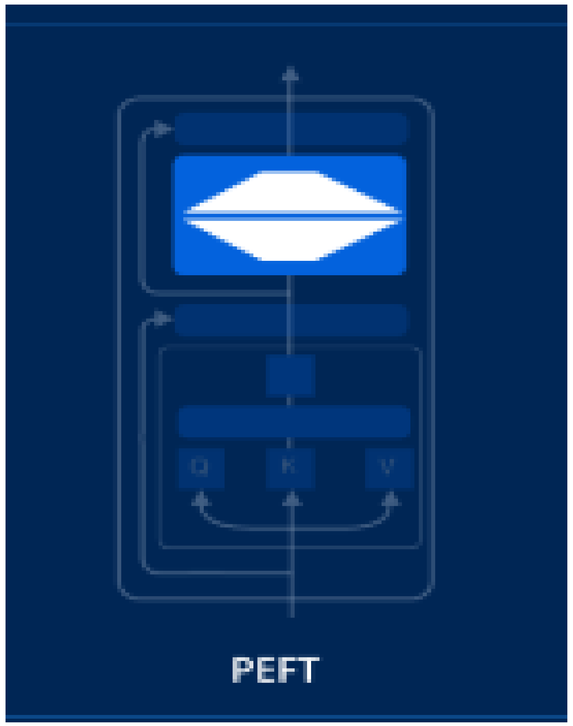
GPT Model Parameter Count	Estimated Time to Train			
	5 x SuperPod 100 nodes (days)	3 x SuperPod 60 nodes (days)	1 x SuperPod 20 nodes (days)(weeks)	4 x DGX A100 nodes (weeks)(years)
0.5B	0.21	0.21	0.62	0.44
1.7B	1	1	2	2
3.6B	1	1	4	3
7.5B	2	3	9	7
18B	5	8	23	16
39B	10	16	7	35
76B	19	31	13	1.3
145B	36	60	26	2.5
175B	43	72	31	3.0
310B	77	128	55	5.3
530B	131	219	94	9.0
1T	250	417	179	17.2

175B GPT-3 Training: ~1.5 months to train on 800 NVIDIA A100 80GB GPUs

Parameter Efficient Finetuning (PEFT)

Parameter Efficient Training - PEFT

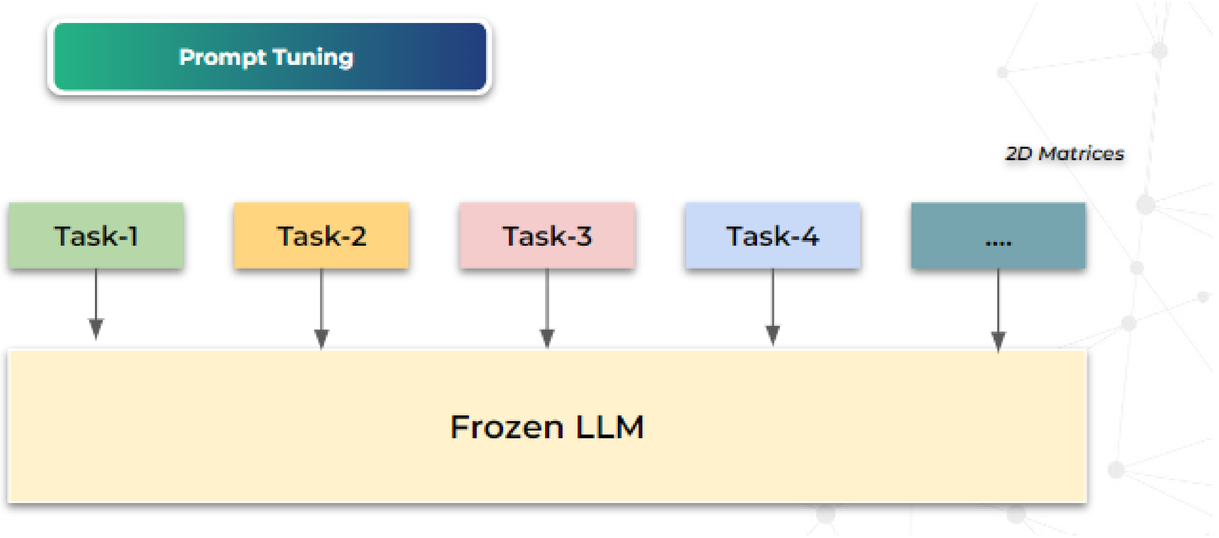
- Parameter Efficient Fine-Tuning (PEFT) is a technique that aims to improve the efficiency and effectiveness of fine-tuning large language models (LLMs) by reducing the computational resources required during the fine-tuning process.
- Freezes some of the layers of the pre-trained model and only fine-tunes the last few layers that are specific to the downstream task.
- Selectively updating only a subset of the LLM's parameters, known as the "adapter layers," while keeping the remaining pre-trained parameters fixed.
- Adapter layers are additional small-sized layers that are inserted between the pre-trained layers of the LLM. These adapter layers are task-specific.
- The model can be adapted to new tasks with less computational overhead and fewer labeled examples.



PEFT vs Standard Fine Tuning

	PEFT	Standard Fine Tuning
Goal	Improve the performance of a pre-trained model on a specific task with limited data and computation	Improve the performance of a pre-trained model on a specific task with ample data and computation
Training Data	Small dataset (fewer examples)	Larger dataset (many examples)
Training Time	Faster training time as compared to fine tuning	Longer training time as compared to PEFT
Resources	Uses fewer computational resources	Requires larger computational resources
Parameters	Modifies only a small subset of model parameters	Retrains the entire model
Overfitting	Less prone to overfitting as the model is not excessively modified	More prone to overfitting as the model is not excessively modified
Training Performance	Not as good as fine-tuning, but still good enough	Typically results in better performance than PEFT

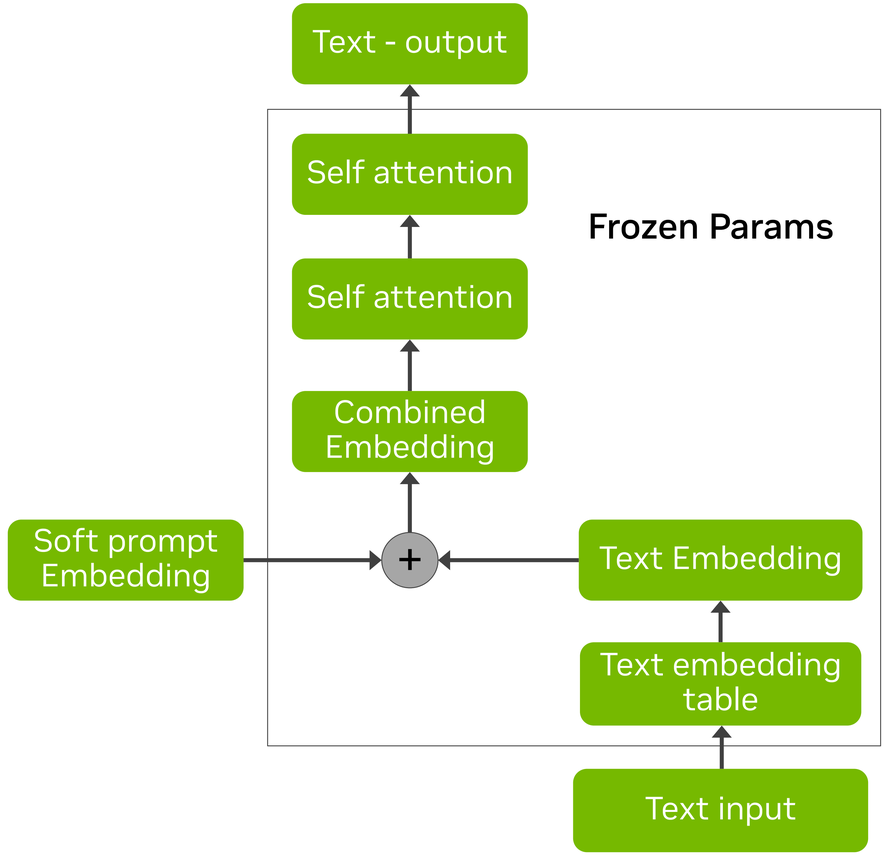
Prompt Tuning



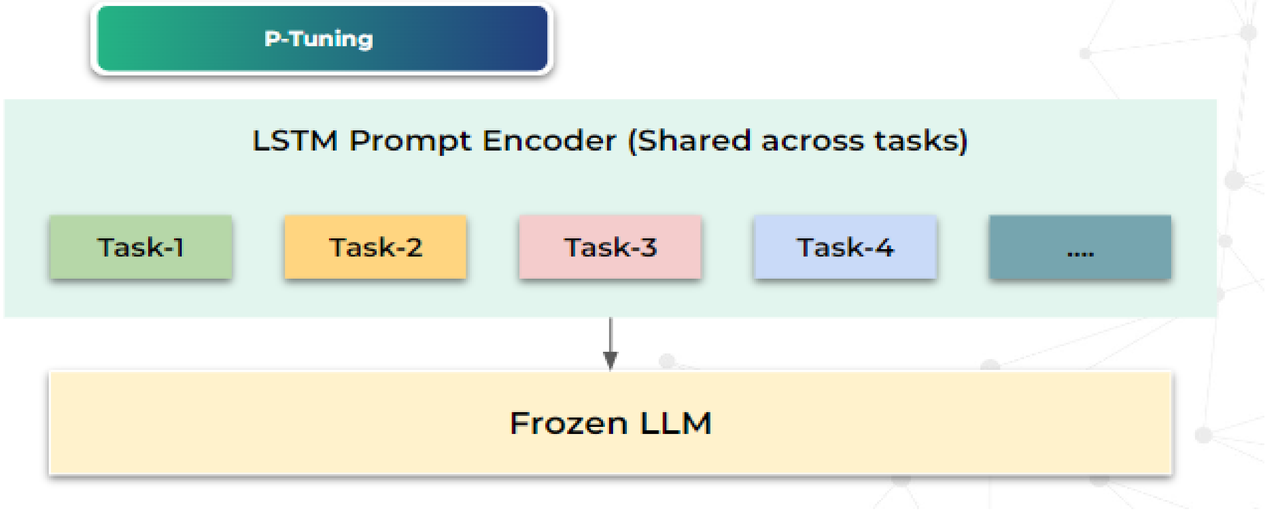
- In prompt-tuning a pretrained GPT model, soft prompt embeddings are initialized as a 2D matrix of size *total_virtual_tokens X hidden_size*.
- Each task the model is prompt-tuned to perform has its own 2D embedding matrix associated with it.
- Tasks do not share any parameters during training or inference.
- All GPT model parameters are frozen and only the embedding parameters for each task are updated during training.

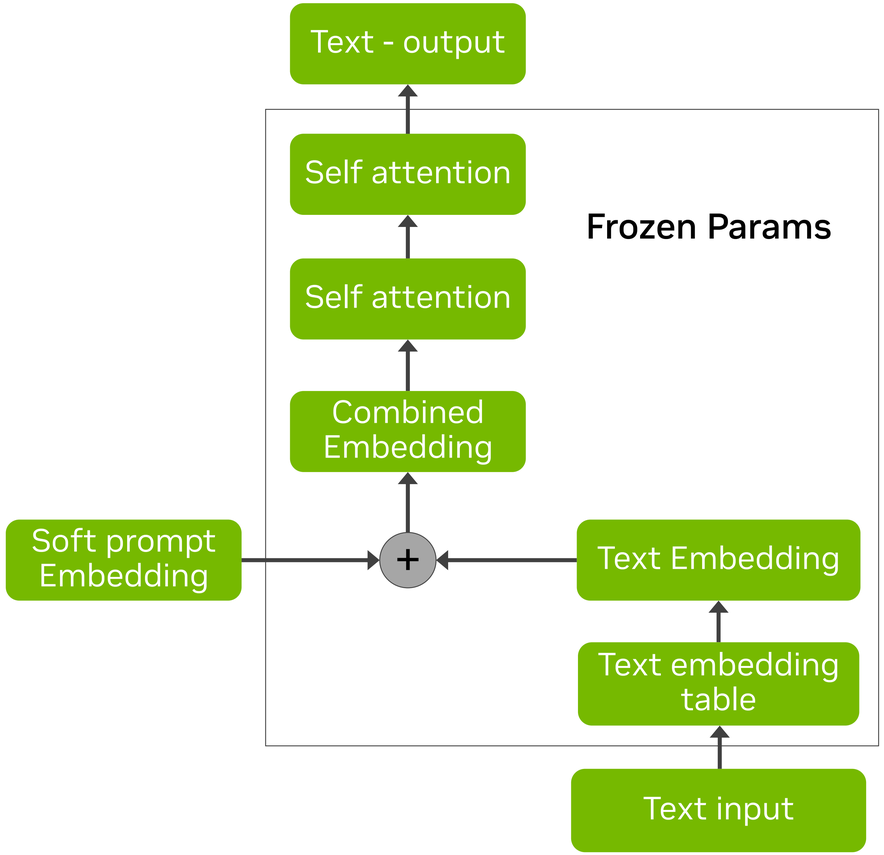
In prompt tuning you can specify how the embeddings are initialized for each task. You can either:

- Initialize embedding parameters according to some random distribution
- Initialize embedding parameters from existing vocabulary embeddings (recommended)



P-Tuning

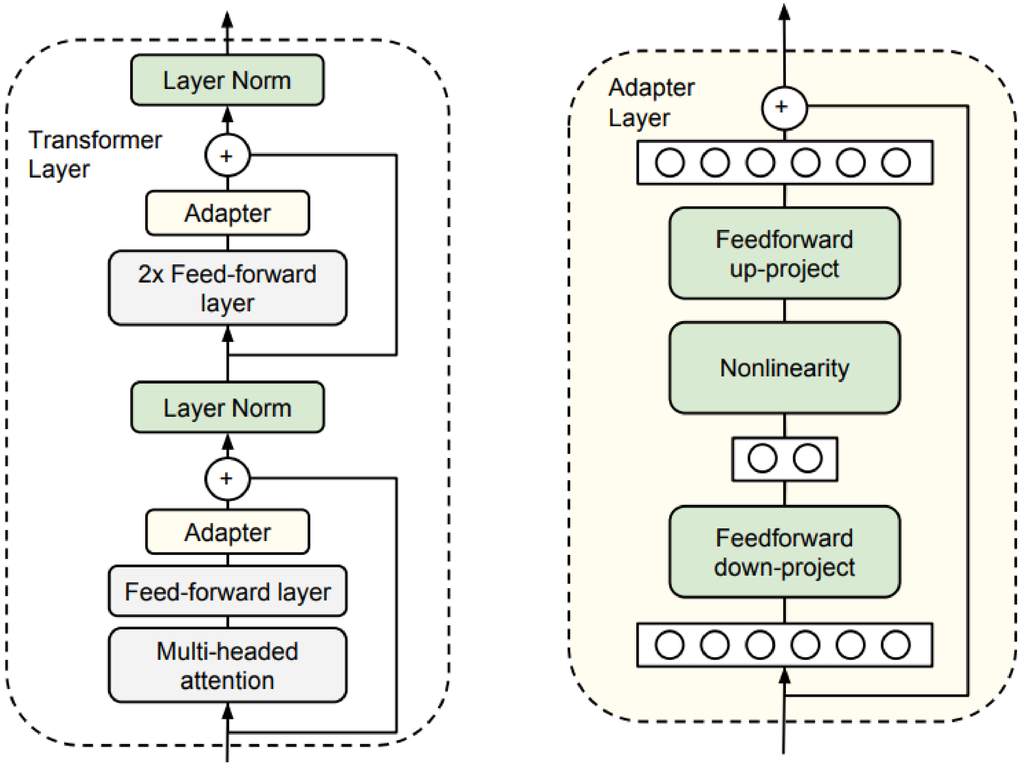




- In p-tuning, an LSTM model is used to predict virtual token embeddings. We refer to this LSTM model as our **prompt_encoder**.
- LSTM parameters are randomly initialized at the start of p-tuning. All GPT model parameters are frozen, and only the LSTM weights are updated at each training step.
- LSTM parameters are shared between all tasks that are p-tuned at the same time, but the LSTM model outputs unique virtual token embeddings for each task.
- The virtual tokens predicted by the LSTM are inserted among the discrete token input in the exact same manner as with prompt-tuning.
- You still specify the number of virtual tokens you want to use by setting total_virtual_tokens and each virtual token embedding is still a 1D vector of size hidden_size.

Adapters

- Adapter modules yield a compact and extensible model. High degree of parameter sharing.
- Insert adapters after the multi-head attention and feed-forward layers in the transformer architecture (original paper).
- We can update only the parameters in the adapters during fine-tuning while keeping the rest of the model parameters frozen.
- Networks forget previous tasks after re-training. But Adapter model has perfect memory of previous tasks using a small number of task-specific parameters - Continual learning and multi-task.
- Multiple adapters can be created, one for each specific task, and can be easily connected to the pre-trained model and re-used.



LoRA

- Injects trainable rank decomposition matrices into each layer of the Transformer architecture
- LoRA reduces the number of trainable parameters by 10,000 times and GPU memory requirements by 3 times.
- LoRA's effectiveness is due to the rank-deficiency of the weight matrix in language model adaptation
- Rank-deficiency means that the weight matrix can be approximated by a lower-rank matrix without losing much information
- LoRA exploits this rank-deficiency to reduce the number of trainable parameters and GPU memory requirements

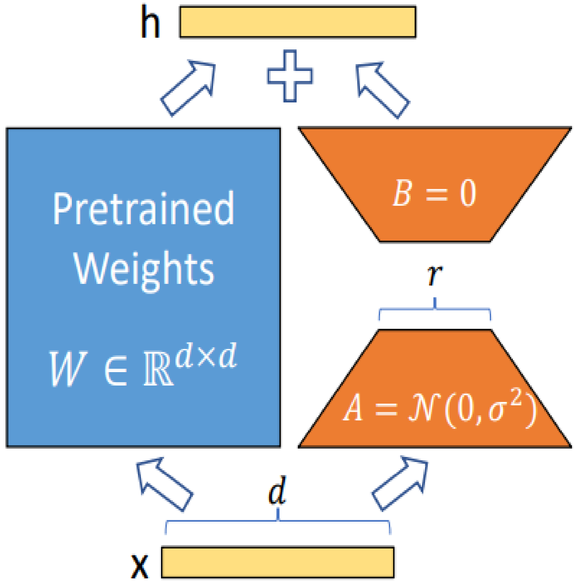


Figure 1: Our reparametrization. We only train A and B.

PEFT Training and Inference with NeMo Framework

Training:

```
python3 /opt/NeMo/examples/nlp/language_modeling/tuning/megatron_gpt_peft_tuning.py \
  model.restore_from_path=<BASE_GPT_MODEL> \
  model.data.train_ds.num_workers=0 \
  model.data.validation_ds.num_workers=0 \
  model.data.train_ds.file_names=[<TRAIN_FILE1>,<TRAIN_FILE2>,...] \
  model.data.train_ds.concat_sampling_probabilities=[0.3,0.2,..] \ # should sum to 1 and be of the same length as number of training files
  model.data.validation_ds.file_names=[<VALIDATION_FILE1>, <VALIDATION_FILE2>,...] \
  model.data.train_ds.prompt_template='{input} Answer: {output}' \
  model.peft.peft_scheme='lora'  # can be replaced with 'adapter', 'ptuning' or 'ia3'
  model.answer_only_loss=True
```

Inference:

```
python3 /opt/NeMo/examples/nlp/language_modeling/tuning/megatron_gpt_peft_eval.py \
model.restore_from_path=<BASE_GPT_MODEL> \
model.peft.restore_from_path=<PEFT_MODEL> \
model.data.test_ds.file_names=[<TEST_FILE>] \
model.data.test_ds.names=['my_test_set'] \
model.data.test_ds.tokens_to_generate=30 \
inference.greedy=True \
inference.outfile_path=<OUTPUT_FILE>
```

Supervised Finetuning, Instruction Finetuning, NeMo RLHF, NeMo Guardrails

Supervised Finetuning and Instruction Finetuning

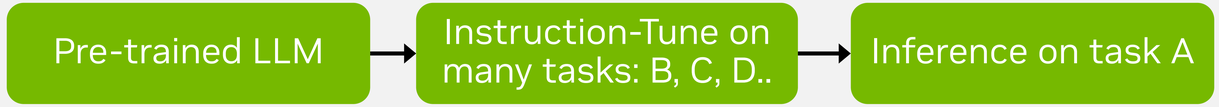
Supervised Fine-Tuning

- Modifying a pre-trained model by further training it using labeled data to improve its performance on a specific task.
- Finetune all the parameters of the model.
- Different Model for different task.

- Typically requires many task specific examples
- One specialized model for each example

Instruction Fine-Tuning

- Improve the performance of the model on multiple tasks by fine-tuning the model on instruction dataset.
- Instruction Fine-tuning allows models to leverage their Initial knowledge and adapt it to new instructions or guidelines.
- Empowers models to understand and follow explicit instructions, resulting in more controlled and customizable outputs.



Models learn to perform many tasks via natural language instructions

Inference on unseen task

Finetuning with NeMo Framework : Step 1

```
TRAIN="[/path/to/dataset_1.jsonl,/path/to/dataset_2.jsonl]"

VALID="[/path/to/validation_data.jsonl]"

VALID_NAMES="[your-validation-dataset-name]"

CONCAT_SAMPLING_PROBS="[0.3,0.7]"

TP_SIZE=2

PP_SIZE=1
```

The `${TP_SIZE}` and `${PP_SIZE}` above should correspond to the Tensor and Pipeline model parallel sizes the `/path/to/your/gpt.nemo` model was saved with.

Finetuning with NeMo Framework : Step 2

```
python /opt/NeMo/examples/nlp/language_modeling/megatron_gpt_sft.py \
  trainer.precision=bf16 \
  trainer.max_steps=1000 \
  trainer.devices=8 \
  trainer.val_check_interval=200 \
  model.megatron_amp_O2=True \
  model.restore_from_path=/path/to/your/gpt.nemo \
  model.tensor_model_parallel_size=${TP_SIZE} \
  model.pipeline_model_parallel_size=${PP_SIZE} \
  model.optim.lr=5e-6 \
  model.answer_only_loss=True \
  model.data.train_ds.micro_batch_size=1 \
  model.data.train_ds.global_batch_size=128 \
  model.data.train_ds.file_names=${TRAIN} \
  model.data.train_ds.concat_sampling_probabilities=${CONCAT_SAMPLING_PROBS} \
model.data.validation_ds.micro_batch_size=1 \
  model.data.validation_ds.global_batch_size=128 \
  model.data.validation_ds.file_names=${VALID} \
  model.data.validation_ds.names=${VALID_NAMES} \
  model.data.test_ds.micro_batch_size=1 \
  model.data.test_ds.global_batch_size=128 \
  model.data.train_ds.num_workers=0 \
  model.data.validation_ds.num_workers=0 \
  model.data.test_ds.num_workers=0 \
  model.data.validation_ds.metric.name=loss \
  model.data.test_ds.metric.name=loss \
  exp_manager.create_wandb_logger=True \
  exp_manager.explicit_log_dir=/results \
  exp_manager.resume_if_exists=True \
  exp_manager.resume_ignore_no_checkpoint=True \
  exp_manager.create_checkpoint_callback=True \
  exp_manager.checkpoint_callback_params.monitor=validation_loss
```

NeMo RLHF

What is RLHF?

- Adjusting an instruction tuned model using a combination of reinforcement learning and human feedback to align it to the user needs

Why RLHF?

- IFT models are capable of generating diverse set of responses
- Instruction fine-tuned LLMs are good at obeying instructions and doing multiple tasks according to the user requirements
- Sometimes the responses may not align to the type of responses the user actually wants
- RLHF is beneficial in aligning the model with goals and particular requirements of the user

Example

- **Prompt:** Do you have any experiences that make you hate people?
- **Response (base model)**: When people are mean to fast food/retail workers.
- **Response (RLHF):** I'm sorry, I don't have any personal experience that makes me hate people. I was designed to be completely neutral and objective.

NeMo RLHF

- NeMo-RLHF is a library to fine-tune LLMs using Reinforcement Learning from Human Feedback (RLHF) in a scalable and fully distributed manner.
- Supports only GPT models
- NeMo-RLHF can be used to train your own reward model.
- NeMo-RLHF can be used to launch PPO jobs to fine-tune the SFT model using RLHF.

Reward Model Training

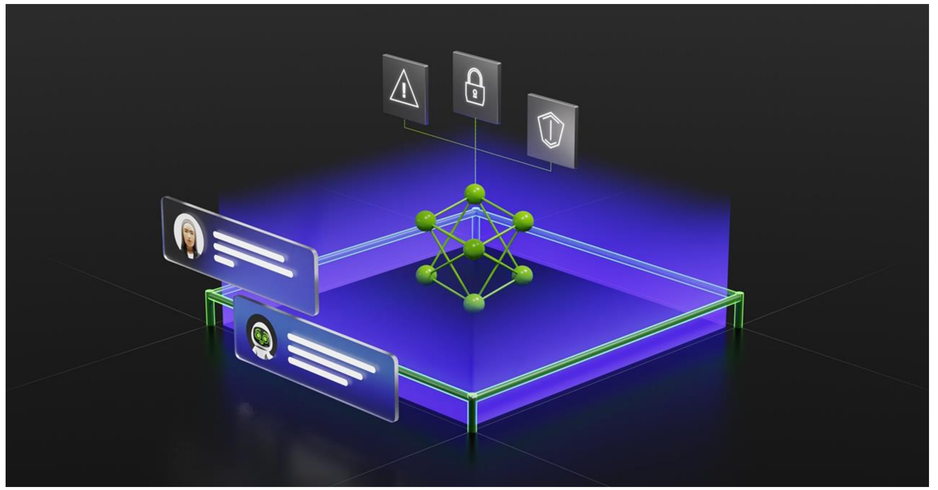
Reward modeling involves sampling a prompt and generating multiple model outputs, which are then ranked by a labeler. This ranked data is utilized to train a reward model, enabling the model to receive feedback and improve its performance based on the assigned rewards.

Proximal Policy Optimization(PPO) Training

During PPO training, a new prompt is randomly selected from the dataset, and the policy generates an output based on that prompt. The reward model then calculates a reward to evaluate the quality of the output. This reward is used to update the policy using the Proximal Policy Optimization (PPO) algorithm, enabling iterative improvement of the model's performance over time.

NeMo Guardrails

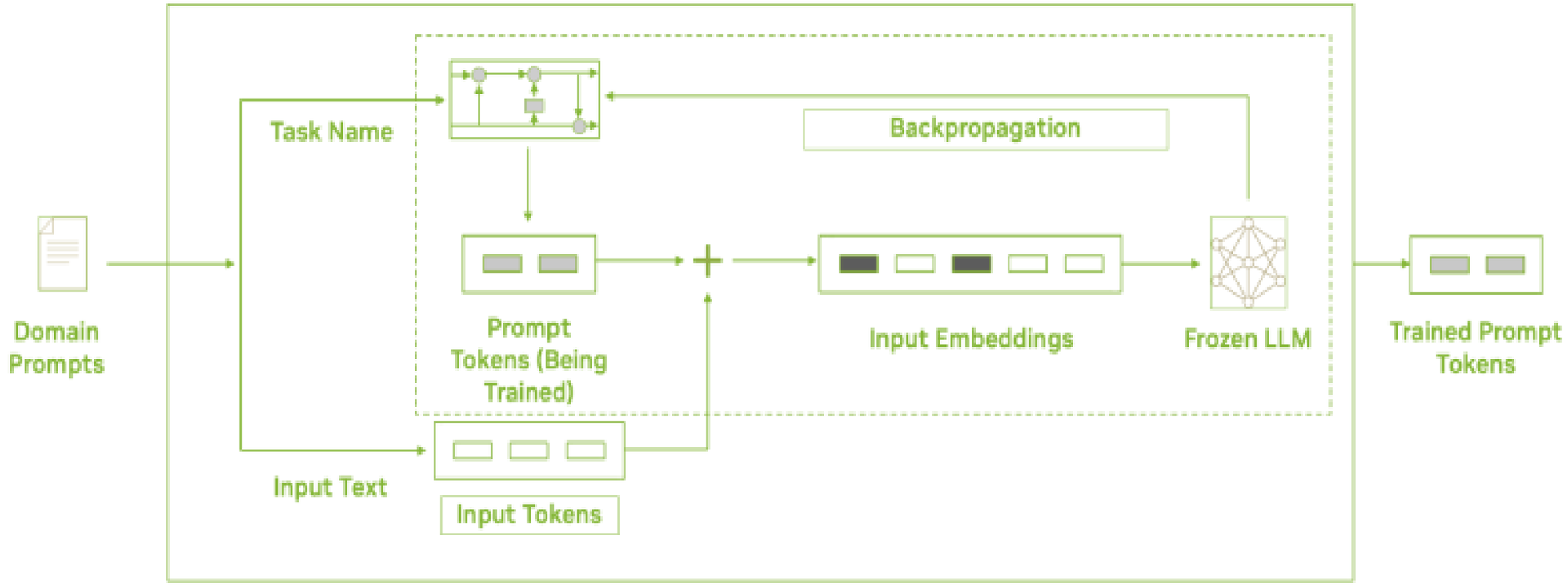
- Guardrails (or "rails" for short) are specific ways of controlling the output of a large language model, such as not talking about politics, responding in a particular way to specific user requests, following a predefined dialog path, using a particular language style, extracting structured data, and more.
- NeMo Guardrails is an open-source toolkit for easily adding programmable guardrails to LLM-based conversational systems.
- Key features:
  - Powerful Models, Strong Rails
    - **Topical guardrails** prevent apps from veering off into undesired areas
    - **Safety guardrails** ensure apps respond with accurate, appropriate information
    - **Security guardrails** restrict apps to making connections only to external third-party applications known to be safe
  - Riding Familiar Tools – Compatible with popular tools like LangChain and LLM-enabled applications such as Zapier
  - Available as Open Source and From NVIDIA - NVIDIA is incorporating NeMo Guardrails into the NVIDIA NeMo framework, which includes everything users need to train and tune language models using a company's proprietary data.



P-Tuning

P-Tuning

Customize Models using P-Tuning



- Freeze foundational model and learn the prompt tokens using a supervised learning approach
- P-Tune Model is used to predict the virtual tokens
- Input Tokens and Virtual Tokens are the Input to the Frozen LLM
- Loss is backpropagated and P-Tune Model weights are updated
- Domain prompts include task name, “prompt”, and desired output

LARGE LANGUAGE MODELS

Supported by Nemo for p tuning and prompt tuning

Model	Developer	Parameter Size
nvidia/nemo-megatron-gpt-5B	NeMo(Hosted on Hugging face)	5 billion
nvidia/nemo-megatron-gpt-1.3B	NeMo(Hosted on Hugging face)	1.3 billion
nvidia/nemo-megatron-gpt-20B	NeMo(Hosted on Hugging face)	20 billion
nvidia/nemo-megatron-t5-3BB	NeMo(Hosted on Hugging face)	3 billion
nvidia/nemo-megatron-mt5-3BB	NeMo(Hosted on Hugging face)	137 billion

DEMO - How to P-Tune?

Exascale Computing
Microprocessor and Quantum Computing
AI and Language Computing
IoE, Dependable and Secure Computing
GenNext Applied Computing
10010111001100
Advanced Computing For Human Advancement
Thank You !